

Meeting 5

Summer 2009 Doing DSP Workshop

Today:

- ▶ Lab exercise 2 exercise VHDL.
- ▶ Linear systems.
- ▶ Transforms.
- ▶ Aliasing.

Education is when you read the fine print. Experience is what you get if you don't.

– Pete Seeger

An option

This Workshop can be used to get EECS 499 credit in the Fall. This would be optional. Five people have expressed an interest.

The Workshop does not have any homework, exams nor any lab reports. Grade would be based on a project. Could be team effort. Base credits would be 2 hours. Could be fewer or more. Details would need to be worked out.

As we work our way through the lab exercises give thought to project possibilities.

Would like to have a poster/demo show-and-tell activity early in the fall term. Would like credit and non-credit projects to participate.

Check out

fpga4fun.com

The posters on display in the EECS atrium today and likely tomorrow. They illustrate the type of research work being done in the department by students and faculty.

Two very good Doing DSP books

I've added Richard Lyons book *Understanding Digital Signal Processing* 2nd edition to the non-required book list on the Workshop web page. Prentice-Hall 2004.

We won't actually be using it, however, it is a very good read and if you have some uncommitted money it is a very worthwhile investment. You can occasionally find it at Borders. Of course, there is always amazon.com.

Lyons has also collected together a set of notes from the IEEE Digital Signal Processing Magazine into the book *Streamlining Digital Signal Processing*, IEEE Press (Wiley-Interscience). These are at a tutorial and applied level. Highly recommended.

Change of emphasis

First three labs will focus on the Spartan-3 FPGA.

Second three labs will focus on the Piccolo.

A few MSP430 boards will be obtained for those who wish to investigate MSP430. No structured lab exercises are planned.

We are giving up breadth on three devices for additional depth on two.

Exercise 1 focus

Exercise 1 — Spartan-3 Starter Board

- ▶ Introduce ISE and Impact.
- ▶ Basic Spartan-3 Starter Board peripherals.
- ▶ VHDL...hardware design language.
- ▶ Constraint file...interface between VHDL and FPGA pins.
- ▶ Practice.

Exercise 2 focus

Exercise 2 – S3SB Analog in, analog out.

- ▶ Use D/A driver and DA2 to generate ramps.
- ▶ Implement synthesizer to generate sine wave D/A out.
- ▶ 1 bit-D/A delta sigma modulator implementation at 0 Hz.
- ▶ Single supply level-shift op-amp circuit.
- ▶ Use drivers to do A/D in to D/A out.
- ▶ Combine A/D-D/A with delta-sigma DAC.

Exercise 3 focus

Exercise 3 – S3SB filter implementation.

- ▶ Single stage of all-pass filter.
- ▶ Cascade of above.
- ▶ Bit-serial multiply-and-accumulate unit.
- ▶ Bit-serial FIR filter implementation.
- ▶ Transfer function measurement.

New. Subject to change.

DACtest0

Generates two ramps, one per PMod D/A for viewing on oscilloscope.

Top level connects the ramp generator with the D/A driver.

A first simple practice test illustrating D/A driver use.

And the DA2 pins are ...

The DA2 block diagram copied from the user's manual did not include actual pin numbers. These are given below.

The DA2 input pins (FPGA side) are:

pin	function
1	SYNC
2	DINA
3	DINB
4	SCLK
5	GND
6	VCC

The DA2 output pins (Top side) are:

pin	function
1	DAC0 output
2	no connection
3	DAC1 output
4	no connection
5	GND
6	VCC

PMod DA2 port description

```
entity pmod_dac0 is
  Port ( go : in  STD_LOGIC;
        da_a : in  STD_LOGIC_VECTOR (11 downto 0);
        da_b : in  STD_LOGIC_VECTOR (11 downto 0);
        pmod : out STD_LOGIC_VECTOR (3 downto 0);
        clk : in  STD_LOGIC);
end pmod_dac0;
```

- ▶ Leading edge of go copies contents of da_a and da_b into the two DACs. There is a latency of about 32 clock cycles.
- ▶ da_a and da_b are unsigned, 0 goes to 0 volts, 4095 goes to V_{cc} .
- ▶ Set of four lines to be connected to PModD module.
- ▶ Max clock is 60 MHz. Counted down by a factor of 2 to clock data. Typically use with 50 MHz or 40 MHz clock.
- ▶ Module is assumed to be in MIB J7 which has name pmod_d.

DAC test 0 top

```
entity DACtest0top is
  Port ( pmod_d : out  STD_LOGIC_VECTOR (3 downto 0);
        led : out  STD_LOGIC_VECTOR (7 downto 0);
        mclk : in  STD_LOGIC);
end DACtest0top;
```

architecture Behavioral of DACtest0top is

```
  signal clk, go : std_logic;
  signal pmod : std_logic_vector(3 downto 0);
  signal ramp_a, ramp_b : std_logic_vector(11 downto 0);
```

begin

```
  pmod_d <= pmod;
  clk <= mclk;
  led <= ramp_b(11 downto 4);
```

```
  dac : entity work.pmod_dac0
  port map(go => go, da_a => ramp_a, da_b => ramp_b,
          pmod => pmod, clk => clk);
```

```
  ramper : entity work.rampgen
  port map(go => go, ramp_a => ramp_a, ramp_b => ramp_b, clk => clk);
```

end Behavioral;

The ramp generator counters

```
entity rampgen is
    Port ( go : out  STD_LOGIC;
          ramp_a : out  STD_LOGIC_VECTOR (11 downto 0);
          ramp_b : out  STD_LOGIC_VECTOR (11 downto 0);
          clk : in  STD_LOGIC);
end rampgen;

architecture Behavioral of rampgen is
    signal a_ramp, b_ramp : std_logic_vector(11 downto 0);
    signal counter : std_logic_vector(5 downto 0); -- 6 bits
begin
    ramp_a <= a_ramp;
    ramp_b <= b_ramp;
    process(clk) is
    begin
        if rising_edge(clk) then
            counter <= counter+1;
            if counter = 0 then      -- divides clock down by 64
                go <= '0';
                a_ramp <= a_ramp+1;
                b_ramp <= b_ramp+3;
            else
                go <= '1';
            end if;
        end if;
    end process;
end Behavioral;
```

DDS for DTMF tone generation

An entity reads slide switches row/column numbers and selects FTV values. Set only one row and one column switch at a time.

An entity divides the 50 MHz clock down to 1 MHz.

Uses two phase accumulators to generate ROM addresses.

ROM contains 256 samples of one period of a sine wave. A block ram was initialized using table values generated using a MATLAB script.

DDS0 (DTMF) project VHDL organization

DDS0top

fetch_FTV	-- select row/column FTV value.
sample_clock	-- generates 1 MHz update clock.
DDS0	-- DDS phase accumulator
DDS1	-- DDS phase accumulator
sine_table	-- sine rom, dual port
pmod_DA2_module	-- DA2 driver VHDL, 2 DACs per DA2
spartan3.ucf	-- project specific UCF file

Basically two simple DDS units with outputs combined at the top level.
Very brute force but very straight forward.

Top level (part 1)

```
begin
  clk <= mclk;
  fetch_ftv : entity work.get_ftv
  port map(
    swt => swt,           -- slide switches
    FTV0 => FTV0,         -- FTV0
    FTV1 => FTV1,         -- FTV1
    clk => clk,
    reset => '0');
  sample_clock : entity fs_clock
  port map (
    fs => fs,
    clk => clk);
  DDS0 : entity work.DDS
  port map (
    FTV => FTV0,          -- FTV value to be used by DDS channel 0
    ROM_address => address_a,
    fs => fs,
    clk => clk,
    reset => '0');
  DDS1 : entity work.DDS
  port map (
    FTV => FTV1,          -- FTV value to be used by DDS channel 1
    ROM_address => address_b,
    fs => fs,
    clk => clk,
    reset => '0');
```

Top level (part 2)

```
sine_table_a : entity work.sine_rom
port map (
    address_a => address_a,
    data_a => sine_a,
    address_b => address_b,
    data_b => sine_b,
    clk => clk);

pmod_DA2_module : entity work.pmod_dac0
port map (
    da_a => sine_a(15 downto 4), -- truncating!
    da_b => sine_b(15 downto 4), -- truncating!
    go => not fs,
    pmod => pmod_d,
    clk => clk);
```

```
end Behavioral;
```

Switches to FTV

architecture Behavioral of get_FTV is

```
constant row1 : std_logic_vector(31 downto 0) := X"00000000";
constant row2 : std_logic_vector(31 downto 0) := X"00000000";
constant row3 : std_logic_vector(31 downto 0) := X"00000000";
constant row4 : std_logic_vector(31 downto 0) := X"00000000";
constant col1 : std_logic_vector(31 downto 0) := X"00000000";
constant col2 : std_logic_vector(31 downto 0) := X"00000000";
constant col3 : std_logic_vector(31 downto 0) := X"00000000";
constant col4 : std_logic_vector(31 downto 0) := X"00000000";
```

begin

```
FTV0 <= row1 when swt(7 downto 4) = "1000" else
        row2 when swt(7 downto 4) = "0100" else
        row3 when swt(7 downto 4) = "0010" else
        row4 when swt(7 downto 4) = "0001" else
        X"00000000";
```

```
FTV1 <= col1 when swt(3 downto 0) = "1000" else
        col2 when swt(3 downto 0) = "0100" else
        col3 when swt(3 downto 0) = "0010" else
        col4 when swt(3 downto 0) = "0001" else
        X"00000000";
```

end Behavioral;

Comments

- ▶ Determining the row and column values are part of the exercise.
- ▶ The two tones could be combined after the DACs using a summing analog op-amp.
- ▶ Will combine ROM outputs digitally. Need to worry about overflow when adding ROM output values together. Easy step is to sign extend values by one bit then sum.

sample clock generator

```
entity fs_clock is
    Port ( fs : out  STD_LOGIC;
           clk : in   STD_LOGIC);
end fs_clock;

architecture Behavioral of fs_clock is

    signal counter : std_logic_vector(5 downto 0);

begin

    tic : process(clk)
    begin
        if rising_edge(clk) then
            counter <= counter-1;
            fs <= '0';
            if counter = 0 then
                counter <= "110001"; -- 49
                fs <= '1';
            end if;
        end if;
    end process;

end Behavioral;
```

Basic DDS

```
entity DDS is
    Port ( FTV : in  STD_LOGIC_VECTOR (31 downto 0);
          ROM_address : out std_logic_vector(7 downto 0);
          fs : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          reset : in std_logic);
end DDS;
```

architecture Behavioral of DDS is

```
    signal accumulator : std_logic_vector(31 downto 0);

begin

    ROM_address <= accumulator(31 downto 24); -- top 8 bits

    process(clk, reset)
    begin
        if reset = '1' then
        elsif rising_edge(clk) then
            if fs = '1' then
                accumulator <= accumulator + FTV;
            end if;
        end if;
    end process;

end Behavioral;
```

Completing the DTMF

- ▶ Need to add the two rom values. Have to make sum one bit larger to allow for carry. Have to sign extend the ROM values before adding.

```
sum <= (sine_a(15) & sine_a) + (sine_b(15) & sine_b);
```

- ▶ Connect the top 8 bits of `sum` to one of the DACs.

A/D in to D/A out

- ▶ Place PMod AD1 module in MIB J3 which is `pmod_b` in the ucf file.
- ▶ Sample rate is 1 MHz divided down by a factor set into the slide switches.
- ▶ Can be used to investigate *aliasing*. Set a relatively low sample rate and use a variable oscillator. Oscillator frequencies around $f_s/2$ and f_s are the most interesting.
- ▶ A/D and D/A use the same clock.

A/D-D/A top (mostly)

```
pmod_ad1 <= pmod_b;    -- connect PMod-AD1 module
pmod_d <= pmod_da2;    -- connect PMod-DA2 module
reset <= '0';
```

```
timing_module : entity work.timing
port map(
    strobe => strobe,
    swt => swt,
    clk => clk,
    reset => reset);
```

```
AD_module : entity work.pmod_adc0
port map (
    go => strobe,
    ad_a => ad0,
    ad_b => ad1,
    pmod => pmod_b,
    clk => clk40);
```

```
DA_module : entity work.pmod_dac0
port map (
    go => strobe,
    da_a => ad0,
    da_b => ad1,
    pmod => pmod_da2,
    clk => clk);
```

```
drive_leds : entity work.led_driver
port map (
    sample => ad0,
    leds => led,
    clk => clk);
```

Sample clock generator

```
architecture Behavioral of timing is
    signal ctr : std_logic_vector(13 downto 0) := (others => '0');
    signal count : std_logic_vector(13 downto 0);
    signal local_strobe : std_logic;

begin
    strobe <= local_strobe;

    -- multiply switches by 50 to allow sampling fractions of 1 MHz
    -- 50 = 32+16+2
    -- no check included for swt = 0

    count <= ('0' & swt & "00000") + ("00" & swt & "0000") + ("00000" & swt & '0');

    process(clk)
    begin
        if rising_edge(clk) then
            ctr <= ctr-1;
            local_strobe <= '0';
            if ctr = 1 then
                ctr <= count;
                local_strobe <= '1';
            end if;
        end if;
    end process;
end Behavioral;
```

Comments

Need to be careful with the setting of the slide switches.

Value of 1 gives sample clock of 1 MHz.

Value of N gives sample clock of $1/N$ MHz.

Setting a low sample rate allows easy investigation of aliasing. See what happens when the signal generator frequency is in the vicinity of $1/(2N)$ MHz and when in the vicinity of $1/N$ MHz.

One-bit DAC

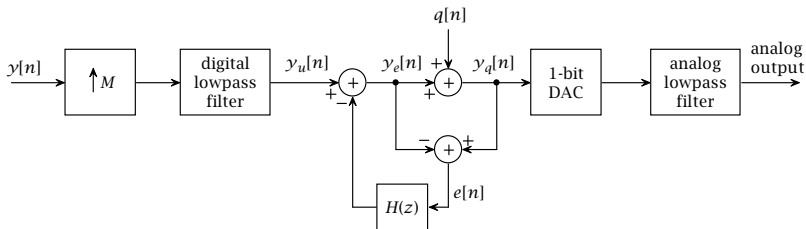
Basic idea is to generate a pulse train whose average value varies with the amplitude of a series of digital inputs. Then lowpass filter.

The resolution of the pulse widths will depend upon the clock rate and the register sizes used.

A delta-sigma modulator is used to control the pulse sizes and transition times to minimize the low frequency noise to the detriment of the high frequency noise.

The high frequency noise is easily attenuated using a lowpass filter.

Delta-Sigma D/A converter block diagram



Input to output TF

$$Y_e(z) = Y_u(z) - H(z)E(z)$$

$$Y_q(z) = Y_e(z)$$

$$E(z) = Y_q(z) - Y_e(z)$$

From this set of equations it is seen that the transfer function between $Y_u(z)$ and $Y_q(z)$ equals 1.

$$Y_q(z) = Q(z) + Y_e(z)$$

$$E(z) = Y_q(z) - Y_e(z)$$

$$Y_e(z) = -H(z)E(z)$$

Quantization noise to output TF

Solving

$$\begin{aligned}E(z) &= -\frac{Y_e(z)}{H(z)} \\Y_e(z) &= \frac{H(z)Y_q(z)}{H(z) - 1} \\ \frac{Y_q(z)}{Q(z)} &= 1 - H(z)\end{aligned}$$

In many texts $H(z) = z^{-1}$. I'm not sure that this is what is used in practice. For this $H(z)$ we have

$$E(z) = 1 - e^{-j2\pi f/f_s}$$

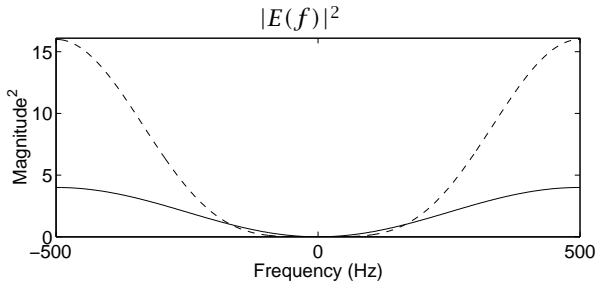
giving

$$|E(f)|^2 = 4 \sin^2(\pi f/f_s).$$

Two useful $H(z)$

The system performance can be improved by replacing the single z^{-1} stage by more sophisticated filter. A filter that has a high pass transfer function will provide improved performance. One that I found in an article has transfer function

$$z^{-1}(2 - z^{-1}).$$



Solid line is for $H(z) = z^{-1}$. Dashed line is for $H(z) = z^{-1}(2 - z^{-1})$.
 $f_r = 1000$.

Xilinx LogiCore Delta-Sigma

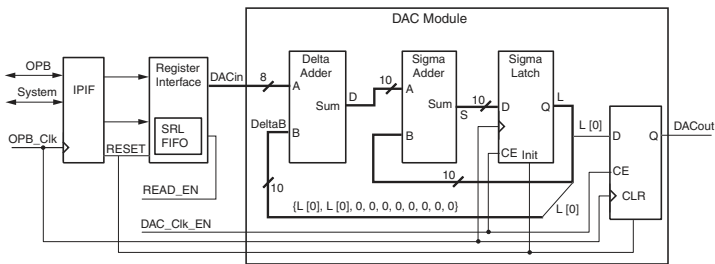


Figure 2: OPB Delta-Sigma DAC Internal Block Diagram

Essentially as the “theoretical” model but modified for use with positive valued data.

Simple RC lowpass filter used to remove the high frequency content.

Comments

- ▶ Switching waveforms (on the board).
- ▶ Xilinx's sign extension and making subtractors into adders.
- ▶ Wasn't able to make the alternative $H(z)$ to work.
- ▶ Design gives 8 effective bits resolution at output.
- ▶ Encountered "surprises".

The A/D-D/A code allows use of an sample rate that is 1 MHz or a submultiple. It is interesting to observe the output using 1 MHz as a function of input sine wave frequency and also using 1/255 MHz sample rate.

For 1 MHz sample rate my implementation works reasonably well up to about 400 Hz.

Doing a good delta-sigma implementation or understanding why one can't would be a good one or two person project.

Changing stride

Next week's exercise will involve implementing a finite impulse response (FIR) filter and one or two infinite impulse response (IIR) filters.

Filters are typically characterized by their effect on sine waves at various filters. Both the effect on amplitude and phase are generally of interest.

Before we start digging into filter theory we need to lay some ground work. More (and probably more lucid) information is contained in the two books on the Workshop CD.

Linear systems

Given two time functions $x_1(t)$ and $x_2(t)$ and a function $h(\cdot)$ (system) such that

$$y_1(t) = h[x_1(t)] \text{ and } y_2(t) = h[x_2(t)]$$

then the system is linear if and only if

$$ay_1(t) + by_2(t) = h[ax_1(t) + bx_2(t)].$$

This leads to the principle of superposition. We can decompose a signal into components, solve for the responses to the individual components and then construct the overall response by adding up the individual responses.

Nonlinear systems are not easy to work with.

Stable, time invariant, causal systems

We say that a system is *stable* if for all bounded inputs the system's output is bounded.

We say that $h(\cdot)$ is *time-invariant* if for $y(t) = h[x(t)]$ we have $y(t - \tau) = h[x(t - \tau)]$.

We say that a system is *causal* if the output never precedes the input.

We will restrict our attention to linear, stable, time-invariant, causal systems.

Continuous time spectra

Fourier transform:

$$X(f) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi ft} dt \quad \text{where } -\infty < f < +\infty,$$

$$x(t) = \int_{-\infty}^{+\infty} X(f)e^{j2\pi ft} df \quad \text{where } -\infty < t < +\infty.$$

Fourier series:

$$c_n = \frac{1}{T} \int_{t_1}^{t_2} x(t)e^{-j2\pi n/(t_2-t_1)} dt, \quad \text{where } -\infty < n < +\infty,$$

$$x(t) = \sum_{n=-\infty}^{+\infty} c_n e^{j2\pi n/(t_2-t_1)} \quad \text{where } t_2 \leq t < t_1.$$

Some restrictions apply.

Discrete Fourier Transform

Discrete Fourier Transform:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \quad \text{where } k = 0, 1, 2, \dots, N-1.$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k]e^{j2\pi kn/N} \quad \text{where } n = 0, 1, 2, \dots, N-1.$$

Do any restrictions apply?

One can move the $1/N$ around or even use $1/\sqrt{N}$ on both.

The z -transform

The z -transform of a discrete set of values, $x(n)$, $-\infty < n < \infty$, is defined as

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n}$$

where z is complex valued. z can be written in polar form as

$$z = re^{j\theta}.$$

r is the magnitude of z and θ is the angle of z . When $r = 1$, $|z| = 1$ is the unit circle in the z -plane.

For causal waveforms (that start at $n = 0$),

$$X(z) = \sum_{n=0}^{\infty} x(n)z^{-n}$$

The inverse z -transform

$$x(n) = \text{ZT}^{-1}[X(z)] = \frac{1}{2\pi j} \oint_C X(z) z^{n-1} dz$$

Some methods of computation:

- ▶ Long division method.
- ▶ Partial fraction expansion method.
- ▶ Use of residues.

See Proakis or a similar text (or Wikipedia?) for details.

Uniform sampling at rate f_s

We can describes angles in the z -plane as

$$\theta = 2\pi f / f_s, \quad \text{where } -f_s/2 \leq f < f_s/2.$$

Then

$$X(z) = \sum_{n=0}^{\infty} x(n/f_s) r e^{-2\pi n f / f_s}.$$

If we restrict ourselves to the unit circle then

$$X(z) = \sum_{n=0}^{\infty} x(n/f_s) e^{-2\pi n f / f_s}.$$

Why would we want to do so? It's useful.

Why use transforms?

The waveform $y(t)$ obtained by processing a waveform, $x(t)$, by a LTIC system having “impulse” response, $h(t)$, can be written as

$$y(t) = \int_0^t x(\tau)h(t - \tau)d\tau .$$

In terms of the transforms of $x(t)$, $h(t)$ and $y(t)$,

$$Y(f) = H(f)X(f) .$$

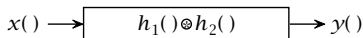
- ▶ It is often easier to think of the effects of LTIC in the (frequency) domain than in the time domain.
- ▶ It is sometimes easier to operate on a waveform in the transform domain than it is in the time domain. In spite of the computational costs of going between domains.

Discrete time transforms

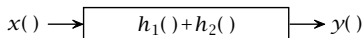
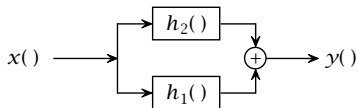
The z -transform will be used to model filter transfer functions in the frequency domain.

The DFT will be used as a computational tool for implementing filters (overlap-and-add algorithm) and for visualizing spectra.

LTI system connections



cascade connection



parallel connection

Waveform spectra

A waveform's power distribution as a function of frequency.

A real valued waveform must have a spectrum that is conjugate symmetric around 0 Hz.

A imaginary valued waveform is not so restricted.

Obviously, real valued waveforms exist only because imaginary valued waveforms exist ;).

$$\text{For example, } \cos(2\pi ft) = \frac{e^{j2\pi ft}}{2} + \frac{e^{-j2\pi ft}}{2}.$$

Delta functions

$$\text{Kronecker delta function: } \delta[n] = \begin{cases} 1 & : n = 0 \\ 0 & : n \neq 0 \end{cases}$$

$$\text{Dirac delta function: } \delta(x) = \begin{cases} +\infty & : x = 0 \\ 0 & : x \neq 0 \end{cases}$$

$$\text{where } \int_{-\infty}^{\infty} \delta(x) dx = 1.$$

$$\text{Sampling: } \int_{-\infty}^{\infty} f(x) \delta(x - a) dx = f(a).$$

Use context to determine whether δ is Kronecker or Dirac.

A touch of reality

A complex number $z = x + jy$ where $j = \sqrt{-1}$ can be thought of as a number pair of reals, $z = (x, y)$ with well defined rules of manipulation. For example for $z_0 = (a, b)$ and $z_1 = (c, d)$

$$z_0 + z_1 = (a + c, b + d)$$

$$z_0 * z_1 = (ac - bd, ad + bc)$$

The rules correspond to those of working with vectors in the plane. The value j is a handy bookkeeping artifice.

Alternatively we can work in terms of polar coordinates:

$$z = r^{j\theta} = (r, \theta).$$

$$z_0 + z_1 = \text{doesn't fit in space available}$$

$$z_0 * z_1 = (r_0 r_1, \theta_0 + \theta_1)$$

Of course, the above values can also be functions of time.

Simply bandlimited waveforms

Lowpass: Negligible energy ($X(f) = 0$) for all $|f| > B$. Single sided bandwidth is B .

If sampled at $f_s > 2B$ can “exactly” reconstruct.

Bandpass: Negligible energy outside of a band, $B = f_2 - f_1$ not containing 0 Hz.

If sampled at $f_s > 2B$ can “exactly” reconstruct. This needs to be done very carefully, not all f_s and B values necessarily work easily.

Note that for bandpass waveforms this is not necessarily $f_s > 2f_2$!

Frequency shifting

Consider

$$s(t) = a(t) \cos[2\pi f_c t + \theta(t)] = \frac{a(t)}{2} \left\{ e^{j[2\pi f_c t + \theta(t)]} + e^{-j[2\pi f_c t + \theta(t)]} \right\}$$

Multiplying $s(t)$ by $e^{-2\pi f_d t}$ gives

$$s(t)e^{-2\pi f_d t} = \frac{a(t)}{2} \left\{ e^{j[2\pi(f_c - f_d)t + \theta(t)]} + e^{-j[2\pi(f_c + f_d)t + \theta(t)]} \right\}$$

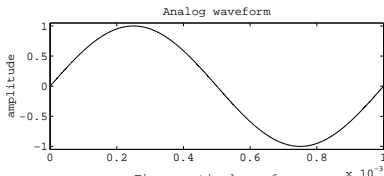
The spectrum is shifted left by f_d Hz. If it should happen that $f_d = f_c$ then

$$s(t)e^{-2\pi f_c t} = \frac{a(t)}{2} \left\{ e^{j[2\pi\theta(t)]} + e^{-j[4\pi f_c t + \theta(t)]} \right\}$$

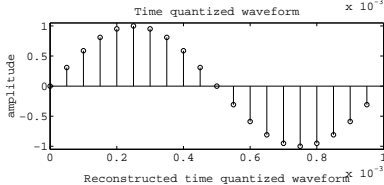
If we can filter out the energy around $-2f_c$ then

$$s(t)e^{-2\pi f_c t} = \frac{a(t)}{2} e^{j2\pi\theta(t)}$$

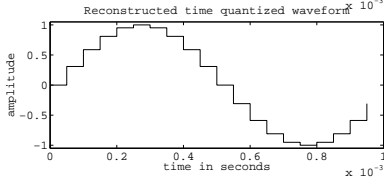
Uniform time quantization & reconstruction



Analog
waveform.



Time
quantized.



Reconstructed.

Aliasing

Sample the waveform $\cos(2\pi f t)$ at rate f_s , $t_n = n/f_s$.

Write $f = \alpha f_s + \Delta$ where α is integer and $0 \leq \Delta < f_s$.

$$\cos(2\pi f n / f_s) = \cos(2\pi n \alpha + 2\pi n \Delta / f_s) = \cos(2\pi n \Delta / f_s).$$

The sample values do not provide any information about the value of α . The sample values for a frequency an integer multiple of f_s from Δ are undistinguishable from the sample values when $f = \Delta$.

The word *alias* means “known by another name”.

When $\alpha \neq 0$ the sample values are said to have been *aliased*.

The range of frequencies aliased to is generally taken to be

$$-f_s/2 \leq \Delta < f_s/2.$$

Units for aliased frequency range

Units *typically* used to describe aliased frequency range:

units	range	limits
nomalized radians	2π	$-\pi \leq \omega < \pi$
Hz	f_s	$-f_s/2 \leq f < f_s/2$
normalized Hz	1	$-1/2 \leq f < 1/2$

Note the inclusion and non-inclusion of the end points.

As practitioners, we will make exclusive use of Hz!

Where does “the” alias land?

Assume the base frequency range is: $-f_s/2 \leq f < f_s/2$.

For a frequency component at $f_c > 0$ find K such that

$$-f_s/2 \leq f_c - Kf_s < f_s/2.$$

The energy at f_c aliases to $f_{ca} = f_c - Kf_s$.

For a frequency component at $f_c < 0$ find K such that

$$-f_s/2 \leq f_c + Kf_s < f_s/2.$$

The energy at f_c aliases to $f_{ca} = f_c + Kf_s$.

We can use aliasing to shift frequencies to “baseband”.

Aliasing demonstration

Using Euler's relation we can write

$$\cos(2\pi ft) = \frac{e^{j2\pi ft}}{2} + \frac{e^{-j2\pi ft}}{2}$$

The movies were generated using a input frequency sweeping from 0 Hz to 24,000 Hz in 30 seconds. The sample frequency was 8 kHz.

The first movie demonstrates what happens when the cosine is sampled.

The second movie demonstrates what happens when only the positive frequency component is sampled.

Both movies show the effects of using a slowly increasing frequency, (linear FM sweep).

swept sinusoid



swept complex exponential



Comments on sampling

The spectrum of an sampled waveform does NOT fold.

Common practice is to make the base frequency range $[-f_s/2, f_s/2)$.

The frequency $f_s/2$ is called the *Nyquist* frequency.

Given a real valued lowpass spectrum with bandwidth, BW the sample frequency equal to $2BW$ is often called the *Nyquist* sample rate.

Reality gets in the way. One should sample at a rate of *at least* two or three times the Nyquist *rate* (not frequency).

Common sample rates:

standard telephone system	8 kHz
wideband telecommunications	16 kHz
home music CDs	44.1 kHz
professional audio	48 kHz
DVD-Audio	192 kHz
instrumentation, RF, video	extremely fast

Some interesting audio frequencies

Many waveforms can have energy beyond a band of interest.

Voice: fundamental around 150 Hz, overtones to about 5 kHz.
 male fundamental about 120 Hz.
 female fundamental about 200 Hz.
 bass low E is 82.4 Hz.
 soprano high C is 1,046.5 Hz.

Piano: 27.5 Hz (A0) to 4816 Hz (C8).

Harmonics may extend frequencies by a factor of 3 to 5 or more.

Normal young adult hearing range is 20 Hz to 20,000 Hz.

Telephone nominally passes range 300 Hz to 3200 Hz.